Paper SM01

R or Python? Dilemma, Dilemma

Elsa Lozachmeur, Idorsia, Basel, Switzerland
Nicolas Dupuis, Sanofi, Basel, Switzerland

ABSTRACT

R™ and Python™ are the new kids in pharma town. These Open Source programming languages seem to be in every discussion lately. We see attempts in our industry to switch to them, replacing SAS®. They indeed come with advantages: licensing cost cut, a large community of users and developers, enabling their fast and impressive development. What are they good at in reality? What are their respective strengths? How do they compare to one another? And even more important, what are their current limitations? What can you do with SAS that you cannot do with R or Python, in your day-to-day Statistical Programmer life? Using our current personal understanding, we will take concrete examples with classic tasks and see how it can be done today with R and Python.

INTRODUCTION

As statistical programmers, we produce datasets (e.g. SDTM, ADaM), tables, listing and figures. In this paper, we will go through some classic tasks done in SAS and show you what it would look like if we had used R or Python instead. As in SAS, the code provided in this paper can be improved or other approaches could be used. R code has been tested in R Studio and Python code in Jupyter Notebook.

WHAT'S R?

Released in 1995, R is a popular, open-source language specialized in statistical analysis that can be extended by packages almost at will. R is commonly used with RStudio, a comfortable development environment that can be used locally or in a client-server installation via a web browser. R applications can also be used directly and interactively on the web via Shiny.

WHAT'S PYTHON?

Python is a fully functional, open-source, interpreted programming language used for general purpose programming that was first released in 1991. It has become an equal alternative to R for data science projects in recent years. Python is particularly well-suited to the Deep Learning and Machine Learning fields and is also practical as statistics software through the use of its ever-growing list of packages, which can easily be installed. A variety of Integrated development environments (IDEs) are available, such as Jupyter, Spyder, and PyCharm. Python is a widely-used language that is also popular in fields like astronomy, finance and web development.

WHY R OR PYTHON?

In our industry we have been using SAS for decades, especially Statistical Programmers when creating reports to the Health Authorities. We are used to it and have grown to be dependent of it for all aspect of our work, so much so, that it took almost 25 years for alternatives to pick up speed. Because of its license structure (cost), its proprietary nature (.sas7bdat) and emergence of Open Source alternatives, we are observing a trend towards decreasing or even eliminating our dependence on SAS. This push is often driven by our younger programmers as R and Python are indeed very popular in universities. New programmers in Data Science are likely to start with R or Python, not SAS, for their trendy activities like data visualization or machine learning. Fields where SAS is lagging way behind.

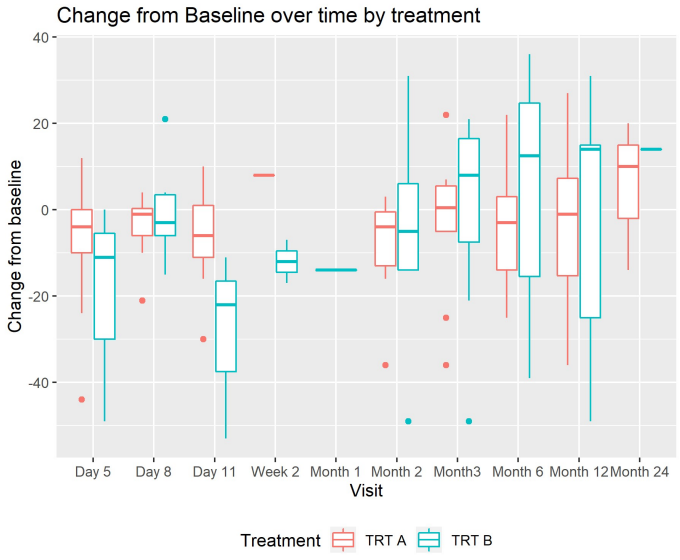
WHAT DO WE WANT TO ACHIEVE?

We took a few classic SAS Statistical Programming tasks and tried replicate them in both R and Python: explore data, derive some ADaM variables, create a table of summary statistics and a boxplot. See below the outputs we will be working on.

Create a Table of summary statistics

		Treatment A			Treatment B		
Time point	Statistics	Baseline	Post-Baseline	Change	Baseline	Post-Baseline	Change
Baseline							
	n	xx			xx		
	Mean	xx.x			xx.x		
	SD	xx.x			xx.x		
	Q1	xx			xx		
	Median	xx			xx		
	Q3	xx			xx		
	Min	xx			xx		
	Max	xx			xx		
Month1							
	n	xx	xx	xx	xx	xx	xx
	Mean	xx.x	xx.x	xx.x	xx.x	xx.x	xx.x
	SD	xx.x	xx.x	xx.x	xx.x	xx.x	xx.x
	Q1	xx	xx	xx	xx	xx	xx
	Median	xx	xx	xx	xx	xx	xx
	Q3	xx	xx	xx	xx	xx	xx
	Min	xx	xx	xx	xx	xx	xx
	Max	xx	xx	xx	xx	xx	xx

Create a Box Plot



R AND PYTHON LIBRARIES

The power of the Open Source lies in its large community of contributors. They create useful libraries that you can freely use. We will use some of these libraries for this paper, see a list in appendix 1.

SOURCE DATA

We will use two SDTM datasets: a DM (demographic data) and a VS (Vital Sign). VS contains common parameters such as: diastolic blood pressure (DBP), Systolic blood pressure (SBP), weight, temperature and height. DBP and SBP are collected at each visit whereas the others at visit 1 and 2.

DM is a SAS dataset and VS dataset a CSV file, so you can see different import methods.

Python

```
# import Demograpic data (format = SAS) into a pandas dataframe
dm = pd.read_sas('C:\\Users\\E0415921\\Documents\\Data Science\\dm.sas7bdat', encoding = "utf-8")

# import Vital Signs (format = CSV) into a pandas dataframe
vs = pd.read_csv('C:\\Users\\E0415921\\Documents\\Data Science\\vs.csv')
```

R

```
#Read source SAS dataset DM
dm <- read_sas("C:\\Users\\xxxxxx\\Documents\\PhUSE\\Paper 2019\\dm.sas7bdat", catalog_file = NULL, encoding =
NULL)
#Read source CSV file
vs <- read.csv("C:\\Users\\xxxxxx\\Documents\\PhUSE\\Paper 2019\\vs.csv")
```

Note that we worked on local files, hence the different paths. We used the same datasets though.

DATA EXPLORATION

Let's see a few commands to explore our data:

Python

```python
# Display a glimpse of the DM dataframe
dm.head()
```

```python
# list all the DM columns
dm.columns
```

```python
# Display distinct values in column RACE. Can be stored in a Python list
list(dm.RACE.unique())
```

```python
# Frequency count on the column RACE. Can be stored in a Pandas series.
dm.groupby(['RACE']).size()
```

```python
# Display some descriptive statistics on the age variable. Can be stored in a Pandas series.
dm["AGEDRV"].describe()
```

```python
# Average value for age, by gender. Can be stored in a Pandas series.
dm.groupby('SEX')['AGEDRV'].mean()
```

```python
# Quick plot with Pandas
dm['AGEDRV'].plot(kind='hist', bins=100, title='Age');
```

R

```r
# Display a glimpse of the DM dataframe
```

```r
tibble(dm)
```

```r
# List all the DM columns
```

```r
head(dm)
```

```r
# Display distinct values in column RACE
```

```r
unique(dm$RACE)
```

```r
# Frequency count on the column RACE
```

```r
table(dm$RACE)
```

```r
# Display some descriptive statistics on the AGE variable
```

```r
summary(dm$AGE)
```

```r
# Average value for AGE, by gender
```

```r
dm2 <- data.table(dm)
dm2[,.(avg=mean(AGE, na.rm = TRUE)), by = SEX]
```

```r
# Quick plot with R
```

```r
g <- ggplot(dm, aes(AGEDRV)) + geom_histogram(binwidth = 0.5) + labs(title = "Age")
g
```

PREPARE THE DATA

Let's do some filtering and merging.

Python

```python
# keep only needed variables from VS
vslight = vs[["NPNO", "VSTESTCD", "VSSTRESN", "VISITNUM", "VSBLFL"]]

# keep only Diastolic blood pressure as per requirements
vslight = vslight[vslight["VSTESTCD"] == "DIABP"]

# merge dm variable
dmlight = dm[["npnon", "TRT01AN"]]
dmlight = dmlight.rename(columns={'npnon': 'NPNO'})
vslight = pd.merge(vslight, dmlight, on = ["NPNO"])

vslight.head()
```

R

```r
# keep only needed variables from VS
```

```r
vs <- vs %>%
  select(NPNO, VSTESTCD, VSSTRESN, VISITNUM, VSBLFL) %>%
  mutate(npnon=as.numeric(NPNO))
```

```r
# keep only Diastolic blood pressure as per requirements
```

```r
vs <- vs %>%
  filter(VSTESTCD == "DIABP")
```

```r
# Merge DM and VS
```

```r
vstrt <- merge(x=vs, y=dm, by="npnon", all.x=TRUE, sort=TRUE)
```

ADAM DERIVATIONS

CREATE CATEGORICAL VARIABLE

Age category is often used to perform sub-group analysis. Let's see how we could derive AGECAT:

Python

```python
# create age categorical variable
dm['agecat'] = pd.cut(dm['AGEDRV'], bins=[18, 40, 60, 80, float('Inf')], labels=['18-40', '40-60', '60-80', '80+'])
```

R

```r
# Create AGECAT variable
```

```r
dm2 <- dm %>%
  select(STUDYID, USUBJID, SUBJID, AGEDRV, COUNTRY, ETHNIC, RACE, SITEID, npnon, TRT01A, TRT01AN) %>%
  mutate(agecat = cut(dmdf$AGEDRV, breaks = c(18,40,60,80,Inf), labels = c("18-40", "40-60", "60-80", "80+")))
```

CREATE BASELINE VALUE

Python

```python
# create baseline values for all parameters
baseline = vslight[vslight["VSBLFL"] ==  "Y"]
baseline = baseline.rename(columns={'VSSTRESN': 'BASELINE'}).drop(["VSBLFL", "VISITNUM", "TRT01AN"], axis = 1)
vslight = pd.merge(vslight, baseline, on = ["NPNO", "VSTESTCD"], how = "left")
```

R

```r
# Create baseline values for all parameters
```

```r
base <- vstrt %>%
  select(NPNO,VSTESTCD,VSSTRESN, VSBLFL) %>%
  filter(VSBLFL == "Y") %>%
  rename.vars(from = "VSSTRESN", to = "BASE") %>%
  select(-VSBLFL)
```

Then merge back to VS dataset to have to only keep those in VS data frame:

```r
vsmerge <- merge(x=vstrt,y=base,by=c("NPNO", "VSTESTCD"),all.y=TRUE,sort=TRUE)
```

CREATE CHANGE FROM BASELINE

Python

```python
# calculate change from baseline
vslight["CHG"] = vslight.apply(lambda row: row['VSSTRESN'] - row['BASELINE'] if row['VSBLFL'] != "Y" else np.nan, axis=1)
```

R

```r
# Calculate change from baseline
```

```r
vschg <- vsmerge %>%
  mutate(AVAL = VSSTRESN) %>%
  mutate(CHG=AVAL - BASE) %>%
  rename.vars(from = "VISITNUM.x", to = "VISITNUM")
```

SUMMARY STATISTICS

In the next step we will derive the summary statistics required as per the table shell:

Python

```python
# statistics on Observed, Baseline, and Chg from bsl
def create_stats(timepoint):
    return vslight.groupby(['TRT01AN', 'VISITNUM', 'VSTESTCD'])[timepoint].describe().reset_index()

obs_stat = create_stats("VSSTRESN") # observed
bsl_stat = create_stats("BASELINE") # baseline
chg_stat = create_stats("CHG") # change from baseline
```

R

```r
#Statistics on Observed, Baseline and change form baseline
# Create a function to get the summary stat for BASE, AVAL and CHG variables
# To debug a function you can use the following: debug(sumstatfun)
```

```r
sumstatfun <- function(VAR)
{  expr <- enquo(VAR)
vschg %>%
  group_by(TRT01AN,VISITNUM,VSTESTCD) %>%
  summarise(n = n_distinct(npnon), mean = mean(UQ(expr)), sd = sd(UQ(expr)), min = min(UQ(expr)),
            max = max(UQ(expr)), median = median(UQ(expr)), na.rm = TRUE)
}
```

```r
# Call the function
```

```r
vs_stat_base <- sumstatfun(BASE)
vs_stat_aval <- sumstatfun(AVAL)
vs_stat_chg <- sumstatfun(CHG)
```

SUMMARY TABLE

First, let's re-arrange our statistics to feed the table

Python

```
# transpose all statistic
chg_stat2 = chg_stat.melt(id_vars = ["VISITNUM", "TRT01AN", "VSTESTCD"])
chg_stat2 = chg_stat2.rename(columns={'value': 'Change', 'variable': 'Statistics'})

obs_stat2 = obs_stat.melt(id_vars = ["VISITNUM", "TRT01AN", "VSTESTCD"])
obs_stat2 = obs_stat2.rename(columns={'value': 'Observed', 'variable': 'Statistics'})

bsl_stat2 = bsl_stat.melt(id_vars = ["VISITNUM", "TRT01AN", "VSTESTCD"])
bsl_stat2 = bsl_stat2.rename(columns={'value': 'Baseline', 'variable': 'Statistics'})

# merge altogether
allstats = pd.merge(bsl_stat2, obs_stat2, on = ["VISITNUM", "TRT01AN", "VSTESTCD", "Statistics"])
allstats = pd.merge(allstats, chg_stat2, on = ["VISITNUM", "TRT01AN", "VSTESTCD", "Statistics"])

# sort and display
allstats = allstats[allstats.Statistics.isin(['count', 'max', 'mean', 'min', 'std'])]
allstats.sort_values(by=["TRT01AN", "VISITNUM", "Statistics"], inplace = True)
allstats.head()
```

Now, let's create the table, manualy since we couldn't find an easy library to help us.

```
# create table from scratch, quick and ugly
items = ["Baseline", "Observed", "Change"]
indentation = [11, 15, 11]

table =  "_____________________________________________________________________________________\n"
table += "                                Treatment A                    Treatment B          \n"
table += "_____________________________________________________________________________________\n"
table += " Visit      Statistics     Baseline   Observed   Change     Baseline   Observed   Change \n"
table += "_____________________________________________________________________________________\n"

for v, visit in enumerate(visits):
    table += " "  + str(visit) + " "  * (indentation [0] -  len (str(visit)))
    for s, stat in enumerate(stats):

        if v == 0:
            if s == 0:
                table += stat + " " * (indentation [1] -  len (str(stat)))
            else:
                table += " " * (indentation [0] + 1) + stat + " " * (indentation [1] -  len (str(stat)))
        else:
            if s == 0:
                table += stat + " " * (indentation [1] -  len (str(stat)))
            else:
                table += " " * (indentation [0] + 1) + stat + " " * (indentation [1] -  len (str(stat)))

        data1 = allstats_A[(allstats_A["VISITNUM"] == visit) & (allstats_A["Statistics"] == stat)]
        data2 = allstats_B[(allstats_B["VISITNUM"] == visit) & (allstats_B["Statistics"] == stat)]

        for item in items:
            try:
                num = data1[item].iloc[0]
                if num != np.nan:
                    display1 = str(round(num, 2)) + " " * (indentation[2] - len(str(round(num, 2))))
                else:
                    display1 = " " * indentation[2]
            except:
                display1 = indentation[2]

            try:
                num = data2[item].iloc[0]
                if num != np.nan:
                    display2 = str(round(num, 2)) + " " * (indentation[2] - len(str(round(num, 2))))
                else:
                    display2 = " " * indentation[2]
            except:
                display2 = " " * indentation[2]

            table += str(display1) + str(display2)

        table += "\n"
    table += "\n"
```

R

In each previous dataset, the statistics are in columns and we would like to have them in lines. To do so, we need to transpose the datasets and merge them all together and assign an ordering variable to be able to display the statistics in correct order. Then add the VISIT variable to get the visit label.

```
# Create a function to transpose stat results and Keep only required Statistics
```

```
transfun <- function(df)
{df %>%
    melt(id=c("TRT01AN","VISITNUM" ,"VSTESTCD"),variable.name = "STATLB", value.name = "STATRES")  %>%
    group_by(VISITNUM,VSTESTCD,STATLB) %>%
    spread(TRT01AN,STATRES, fill = NA) %>%
    filter(STATLB != "na.rm")
}

vs_trans_base <- transfun(vs_stat_base)
vs_trans_aval <- transfun(vs_stat_aval)
vs_trans_chg <- transfun(vs_stat_chg)
```

```
# Rename variables
```

```
vs_trans_base2 <- vs_trans_base %>%
    rename.vars(from = "1", to = "TRTA.BASE") %>%
    rename.vars(from = "2", to = "TRTB.BASE")

vs_trans_aval2 <- vs_trans_aval %>%
  rename.vars(from = "1", to = "TRTA.AVAL") %>%
  rename.vars(from = "2", to = "TRTB.AVAL")

vs_trans_chg2 <- vs_trans_chg %>%
  rename.vars(from = "1", to = "TRTA.CHG") %>%
  rename.vars(from = "2", to = "TRTB.CHG")
```

```
# Merge all statistics together
```

```
vs_stat_ba <- merge(x=vs_trans_base2, y=vs_trans_aval2, by=c("VISITNUM","VSTESTCD","STATLB"), all.x=TRUE, sort=
TRUE)
vs_all_stat <- merge(x=vs_stat_ba, y=vs_trans_chg2, by=c("VISITNUM","VSTESTCD","STATLB"), all.x=TRUE, sort=TRUE
)
```

```
# Create an ordering variables for Stat
```

```
vs_ord <- vs_all_stat %>%
          mutate(ORD = case_when(
            STATLB == "n" ~ 1,
            STATLB == "mean" ~ 2,
            STATLB == "sd" ~ 3,
            STATLB == "median" ~ 4,
            STATLB == "min" ~ 5,
            STATLB == "max" ~ 6)) %>%
         arrange(VSTESTCD,VISITNUM,ORD)
```

```
# Create a list to get visit label
```

```
dt_vs <- data.table(vs_ord)
```

```
# Get value of visitnum
```

```
unique(dt_vs[,VISITNUM])
visit <- data.frame(VISITNUM = c(1, 5, 8, 11, 14, 25, 26, 27, 28, 29, 30),
                    VISIT = c("Baseline", "Day 5", "Day 8", "Day 11", "Week 2", "Month 1", "Month 2", "Month3",
 "Month 6", "Month 12", "Month 24"))
```

```
# Merge back to vs_param to get VISIT variable
```

```
vs_vis <- merge(x=vs_ord, y=visit, by="VISITNUM", all.x=TRUE, sort=TRUE)
vs_vis2 <- unique(vs_vis)
```

```
# Sort input dataset
```

```
diabp <- vs_vis2 %>%
    arrange(VISITNUM, ORD)
```

The package FLEXTABLE used to generate the table. Other packages such as TABLES, EXPSS, RTALBES can also be used to perform this task. In addition to FLEXTABLE package, some functions of the OFFICER package are also used. See Appendix for details on the FLEXTABLE package.
Here is the command to initiate the report:

```
ft <- flextable(diabp,
                col_keys = c("VISIT", "STATLB", "TRTA.BASE", "TRTA.CHG", "TRTA.AVAL", "TRTB.BASE", "TRTB.AVAL",
 "TRTB.CHG"),
                cwidth = .5)
```

Now we will create the upper column header with the appropriate label:

```
ft <- set_header_labels(ft, VISIT= "", STATLB ="", TRTA.BASE = "Treatment A", TRTA.AVAL = "Treatment A" ,TRTA.C
HG = "Treatment A", TRTB.BASE = "Treatment B", TRTB.AVAL = "Treatment B", TRTB.CHG = "Treatment B")
ft <- merge_at(ft, i = 1, j = 3:5, part = "header")
ft <- merge_at(ft, i = 1, j = 6:8, part = "header")
```

Now we are going the create the lower column header with its label:

```
ft <- add_header_row(ft, values = c("Visit", "Statistics", "Baseline", "Observed", "Change", "Baseline", "Obser
ved", "Change"), top = FALSE)
ft <- add_header_lines(ft, values = "Parameter: Diastolic Blood Pressure (mmHg)")
ft <- theme_booktabs(ft)
```

With the following commands, the line below the line “Parameter:…” will be removed , the Treatment header and the results will be aligned and we will merge the cells that contain the visit information to not have the Visit label repeated on each line.

```
ft <- hline_top(ft, j = 1:8, part = "header", border = fp_border(color = "white", width = 0))
```

```
ft <- align(ft, i=1, align = "left", part = "header")
ft <- align(ft, i = 2,j = 3:5, align = "center", part = "header")
ft <- align(ft, i = 2,j = 6:8, align = "center", part = "header")
ft <- align(ft, i=3, j = 3:5, align = "right")
ft <- align(ft, i=3, j = 6:8, align = "right")
ft <- align(ft, j = 3:5, align = "right", part = "body")
ft <- align(ft, j = 6:8, align = "right", part = "body")
```

```
# Merge identical values
```

```
ft <- merge_v(ft, j = "VISIT")
ft <- valign(ft, j=1, valign = "top", part = "body")
```

We can also customize further the layout with the following steps such as format the numeric values of the summary statistics and replace the missing value by 0:

```
colkeys = c("TRTA.BASE", "TRTA.CHG", "TRTA.AVAL", "TRTB.BASE", "TRTB.AVAL", "TRTB.CHG")
ft <- colformat_num(ft, col_keys=colkeys,  digits = 2, na_str="0")
```

change the font and the font size,

```
ft <- font(ft, part = "all", fontname = "Courier")
ft <- fontsize(ft, part = "all", size = 8)
```

add a title,

```
ft <- add_header_lines(ft, values="Table 15.2.3-1-2 Summary statistics of Vital signs by treatment and visit (Safety Set)",
top=TRUE)
```

and finally adjust the cell widths and heights:

```
ft <- autofit(ft)
ft <- width(ft, j = 1, width = .8)
```

Now here are the final steps to generate a word document

```
#setwd("C:/Users/lozacel1/OneDrive - Idorsia Pharmaceuticals Ltd/Documents/PhUSE/Paper 2019")
doc <- read_docx()
doc <- body_add_flextable(doc, value = ft, align = "center")
print(doc, target = "PhUSE_DIABP_SUM_STATS.docx")
```

BOXPLOT

R and Python offer excellent solutions for graphs.

Python

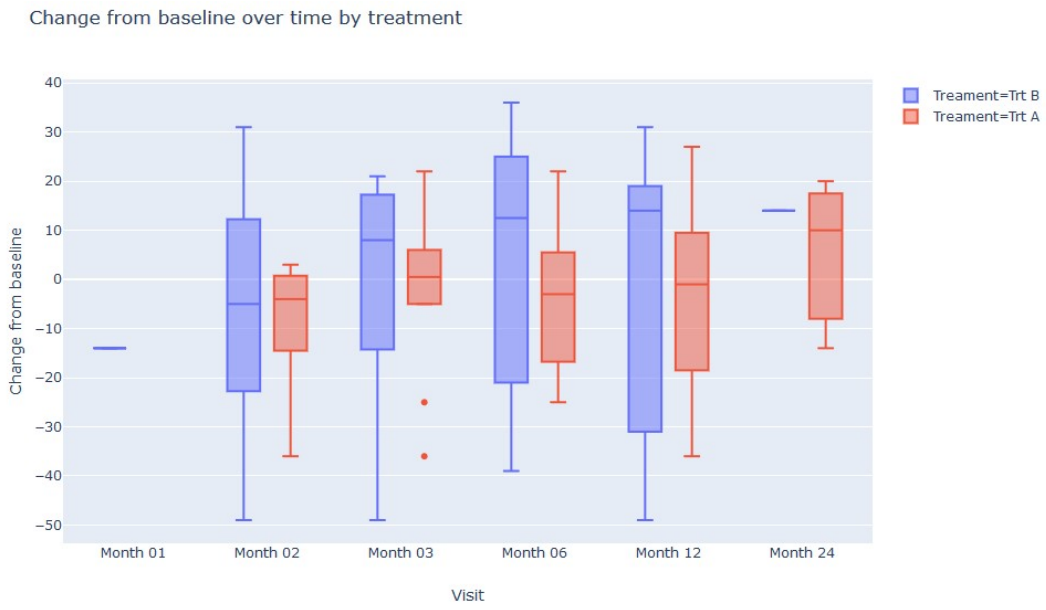
```
# Prepare data for plots
data = vslight[vslight["VISITNUM"] >= 25]
data['VISITNUM'] = data['VISITNUM'].map({30: "Month 24",
                                         29: "Month 12",
                                         28: "Month 06",
                                         27: "Month 03",
                                         26: "Month 02",
                                         25: "Month 01"})

data = data.rename(columns={'TRT01AN': 'Treament'})
data['Treament'] = data['Treament'].map({1: "Trt A",
                                         2: "Trt B"})

data.sort_values(by=["VISITNUM"], inplace = True)
```

```
# Boxplot with Plotly
fig = px.box(data,
             x = "VISITNUM",
             y = "CHG",
             color="Treament",
             title = "Change from baseline over time by treatment" )

fig.update_layout(xaxis = go.layout.XAxis(title=go.layout.xaxis.Title(text="Visit")),
                  yaxis = go.layout.YAxis(title=go.layout.yaxis.Title(text="Change from baseline")))
fig.show()
```



R

First, we keep only the required variables and filter on Diastolic blood pressure parameter.

```
dsplot <- vschg %>%
  select(TRT01AN, TRT01A, NPNO, VISITNUM, VSTESTCD, CHG) %>%
  filter(VSTESTCD == "DIABP", VISITNUM %in% c(25,26,27,28,29,30))

dsplot2 <- merge(x=dsplot, y=visit, by="VISITNUM", all.x=TRUE, sort=TRUE)
```

It is known that reordering groups in a ggplot2 chart can be a struggle. This is because ggplot2 takes into account the order of the factor levels, not the order you observe in your data frame. You can sort your input data frame with sort () or arrange (), it will never have any impact on your ggplot2 output. Here is a workaround:

```
# Re-order VISIT following the value of another column (VISITNUM)
```

```
dsplot3 <- dsplot2 %>%
           arrange(TRT01AN, VISITNUM) %>%
           mutate(VISORD = fct_reorder(VISIT, VISITNUM))
```

```
# Boxplot code
```

```
p <- ggplot(dsplot3, aes(x=VISORD, y=CHG)) +
     geom_boxplot(aes(colour = TRT01A)) +
     theme(legend.position = "bottom", legend.title = element_text()) +
     labs(title="Change from Baseline over time by treatment", x = "Visit", y = "Change from baseline", color =
 "Treatment")
p
```

```
# To produce a word document
```

```
#setwd("C:/Users/lozaceli/Documents/PhUSE/Paper 2019")
docg <- read_docx()
docg <- body_add_gg(docg, value = p, style = "centered")
print(docg, target = "PhUSE_SM01_graphs.docx")
```

COMPARISON

ACHIEVEMENTS

We could do everything we planned. The main challenge was to produce the summary table itself (not the statistics). R doesn't have a defined way of producing reports and to do so, it took time and need some programing investment to learn Flextable package in R. As for Python, it seems to be lacking an easy way to create statistical reports, something like our good old Proc Report. It would be useful to have a user-friendly library to create these tables, without learning templating languages like Jinja2 for example. Instead we did some quick and dirty, unstainable programming. Creating a 'Proc Report' like library in Python would be an interesting project.

As for graphics, R ggplot2 package is probably the most famous, powerful package and very easy to use. For Python, Pandas offers some straight of the box though limited solutions. Matplotlib is much more comprehensive but also harder to learn. Plotly did the trick for us: nice and easy. The visualizations can be breath taking.

Altogether, it took time investment to get familiarity, nothing unexpected of course.

LEARNING CURVE

To be fair, the learning curve may seem quite steep to SAS programmers:

- Python is an easy language to learn but Pandas (any external library as a matter of fact) has its perks, it takes time at first. Debugging can be challenging, StackOverFlow.com will become your best friend.
- For R, you need to know the packages that will meet your needs, seek them out, install them and find out what they can do. There are more than 4300 packages (and the number is growing every day) so it can be difficult and time consuming when searching among them.

STABILITY

So, there is large community developing these Open Source languages and the libraries. Not all groups have a rigorous governance, professional approach to, for example, making backward compatibility a priority. There is no problem for core R and core Python, things can sometimes get messy with some packages. Anyone who has used Pandas saw deprecated APIs, so old code may not work with the current version of Pandas. In that particular case, the developers have announced release soon of version 1 with stable APIs over time, ensuring backward compatibility. We need anyway to learn using virtual environments or containers to encapsulate libraries and ensure reproducibility over time.

IDE

For Statistical Programmers, the quality of the IDE (Integrated Development Environment) is key. RStudio is very popular and there's no direct equivalent for Python. However, RStudio offers already some interoperability with Python and is certainly poised to be fully compatible for Python one day. Alternative solutions like Jupyter Notebook are quite good as well.

PERFORMANCE

In terms of performance, we didn't notice anything wrong but of course we didn't use large amount of data. Benchmarks would be quite an interesting thing to do. SAS is famously efficient when processing data. Performance cannot be the reason why you move away from SAS. Both R and Python are interpreted languages, not compiled and will be slower than SAS. Cython or Julia may become one day interesting alternatives as they are compiled and really fast.

COMPLIANCE

SAS is a validated platform, a requirement in our universe. There are solutions to have validated R platforms (e.g. Mango), we don't have an equivalent solution for Python. As for the Health Authorities, no regulations prohibit the use of R! Its use is at the organization risk, there must be proper validation, documentation and accountability. Results should reproducible and independent of the software used to derive them. You can use R and submit!

DOCUMENTATION

SAS documentation is very good and comprehensive. You'll find countless websites, articles, blogs, videos, trainings on Python and R. The documentation is there and sometimes excellent but doesn't always reach professional quality. The vibrant community makes up for the lack of a hotline. However, users rely on what others put out there about the software. There is a disconnect in the world-wide user group because the developers are so spread out. Packages are not written by the R Development Core-Team therefore they are not always well polished and may have questionable validity or not thoroughly be tested. Support can be also difficult, and R's somehow disorganized documentation and lack of technical support make finding help a challenge. This is a big trade-off for the developer-oriented design of R.

CONCLUSION

We see a trend in the industry to use more and more R for statistical analysis, trying to reduce our SAS footprint. Why R? Probably thanks to statistician's influence. They have been using R for a long time, especially for statistical models (something where Python offers less, today). The latest statistical developments are available in R years I not decades before they are implemented in SAS. It also provides a robust alternative to SAS for everything we need to do. Applications like R Shinny, for dynamic visualizations, are becoming very popular. Is R the way forward? In many industries, data science activities are done with Python and the trend is overwhelming. Python is extremely popular for Machine Learning and AI, it even looks like a monopoly. Thanks to a massive community, the development of Python libraries for data science and analysis is impressive hence why Python is taking over the world. It has yet to conquer Pharma and Statistical Programming, where R is more suited to the classic things we do.

We did not mean to make a choice here. The 3 languages can still live together for now, and sometimes interact with each other (e.g. coding Python stuff in a SAS or in an R program or the other way around). We strongly believe however that Open Source solutions are here to stay. Will this change the way we work? Will we be more efficient? Time will tell. It shouldn't be about replacing a language to do exactly the same, it shouldn't be just about license cost. There are clear advantages for our industry to move towards these new languages, for simple or complex applications. For sure, learning R for statistical analysis or Python for data science/general programming will pay back.

RECOMMENDED READING

https://www.r-bloggers.com/whats-the-best-statistical-software-a-comparison-of-r-python-sas-spss-and-stata/

Python	https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html https://plot.ly/graphing-libraries/ https://jakevdp.github.io/PythonDataScienceHandbook/
R	Book About R: R for Data Science (Garett Grolemund and Hadley Wickham) https://r4ds.had.co.nz/index.html Useful website for R: https://stackoverflow.com/questions/ https://www.r-bloggers.com/ https://www.r-graph-gallery.com/index.html About Flextable package: https://davidgohel.github.io/flextable/articles/overview.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Elsa Lozachmeur: elsa.lozachmeur@idorsia.com
Nicolas Dupuis: ndupuis@protonmail.com

APPENDIX 1: LIST OF LIBRARIES

For R:

```r
library(base)
library(utils)
library(datasets)
library(gdata) #To rename the variable name
library(haven) #To read the sas dataset
library(sas7bdat) #To read the sas dataset
library(reshape2) #To make some changes like transpose
library(plyr) # to load before DPLYR package
library(dplyr) #To subset the dataset
library(magrittr) #to be able to use the pipe operator
library(tidyverse) #to be albe to use select function
library(lubridate) #to work with dates and times (or hms: more for time ? to be investigated)
library(flextable) #to create report
library(officer) #to create report
library(rmarkdown)
library(rtf)
library(data.table)
```

For Python:

```python
# import all needed libraries
%matplotlib inline
from IPython.display import display, HTML

import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
```

APPENDIX 2

The FLEXTABLE package has been build based on the following approach: